Dynamic Resource Allocation using Docker over Cloud for Distributed Computing

Patil Pankaj Shantaram[1] Prof. R. B. Wagh[2]
[1]PG Student [2]Assistant Professor
[1,2]Department of Computer Engineering
[1,2]R. C. Patel Institute of Technology Shirpur, Maharashtra, India

Abstract— Recently, by rapidly development of internet cloud computing is become a new computing model, which also moved computing and data away from desktop and portable PCs into large data centers. It is been observed that cloud computing has many challenges such as poor resource utilization which affects the performance of cloud computing due to the huge amounts of information. Therefore to improve the performance of cloud computing load balancing algorithms are used. By using load balancing algorithms high availability, flexibility, cost reduced and on demand scalability factors are achieved in cloud computing. In such way cloud computing is becoming a modern style of computing globally which is using the power of Internet and wide area network (WAN) to offer resources remotely. Still in the system performance there is some inefficiency and load is imbalance. There are several amount of load balancing and job scheduling algorithms in cloud computing, in this research to improve the performance and efficiency in heterogeneous cloud computing environment a dynamic resource allocation is used over cloud for distributed computing. Also a container algorithm is introduced in this research based on randomization and fuzzy inference algorithm. By considering runtime CPU capacity factor and current resource information several objectives were achieved, such as processing time and average response time of the environment, which further results in improvement in performance of cloud computing environment. In such way heterogeneous environment of cloud computing is providing rapidly and on-demand wild range of service to end users.

Keywords: Wide Area Network (WAN), Big Data, Docker, Cloud Computing

I. INTRODUCTION

A. Introduction of Big Data

In recent years, by development of internet rapidly, cloud computing become a new computing model. In which cloud computing also provides formal resource sharing with middleware's, application development platforms, and business applications. Cloud is consisting of usable and accessible resources which are used for resource utilization. For DNA sequence alignment large and complex amounts of data processing is required and also required targeting and scheduling of resources to solve this problem. It discovers new cloud computing evolution as a distributed systems. Cloud computing is a heterogeneous environment in which Map Reduce platforms, resource allocation are remains a challenge. So, some enhanced architectures are introduced to reduce the computation cost which is consist of Big Data also provides solutions over issue of resource allocation. Cloud computing also offers a rapidly and on-demand wide range of services. In this, Heterogeneous environment means having different kind of hardware characteristics including CPU, memory, storage and other hardware .The business owner can start and expand without invest in the infrastructure with lowering operating and maintenance cost. It has moved computing and data away from desktop and portable PCs, into large data centers. It has the capability to harness the power of Internet and wide area network (WAN) to use resources that are available remotely, thereby providing cost effective solution to most of the real life requirements. The National Institute of Standards and Technology's (NIST) define a Cloud computing as "cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. Load balancing is considered as one of the challenges in cloud computing, it is the major factor to improve the performance of the cloud computing. The current load balancing algorithms in cloud computing environment is not highly efficient. Load balancing in cloud computing environment is very complex task till today, because prediction of user request arrivals on the server is not possible. Each virtual machine has different specification, so it becomes a very difficult to schedule job and balance the load among nodes. Recently, many research works that have been done on load balancing in cloud computing such as Round Robin, Equally Spread Current Execution and Throttled Load Balancing Algorithm. There is other works done using randomization such as ant colony algorithm. In this research a container based algorithm is introduced that takes advantages of both random and fuzzy inference algorithm. The experiments done using cloud server to test the performance of the proposed algorithm in heterogeneous of processors power. The experiments studied the effect of considering the capacity of CPU factor with such algorithm in heterogeneous environment of hosts, and studied the effect of network delay on the containers. The results showed improvements on average response time and on processing time by considering the current resource information and the CPU capacity factor. In recent years, lighter-weight virtualization solutions have begun to emerge as an alternative to virtual machines. Because these solutions are still in their infancy, however, several research questions remain open in terms of how to effectively manage computing resources. One important problem is the management of resources in the event of virtualization. For some applications, overutilization can severely affect performance. Some solutions to this problem are provided by extending the concept of time slicing to the level of virtualization container. Through this approach one can control and mitigate some of the more detrimental performance effects oversubscription. Results show significant improvement over standard scheduling with Docker.

B. Hadoop

Hadoop is the framework which is used for distributed processing and storage of large datasets into cluster. As Hadoop works with the Map Reduce algorithm and get written in java which also known as Apache open source framework provides solutions for Big Data processing and analysis. Figure 1 shows Hadoop architecture which has two major layers namely, Processing or Computation layer consist of Map Reduce which provides the system analysis and Storage layer provides data reliability consist of Hadoop Distributed File System. Hadoop consist of core tasks in which divides data into directories and files then distributed such data for further processing. HDFS is located on top of the local file system which supervises the processing in which blocks are replicated used in handling hardware failure and checking that the code was executed successfully. Hadoop framework is designed for detecting and handling failures at application layer and automatically distributes the data and utilizes parallelism.

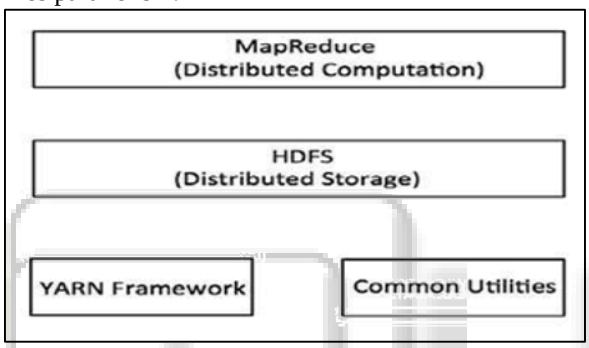


Fig. 1: Hadoop Architecture

C. Docker and Docker Container

Docker is a containerization platform that packages your application and all its dependencies together in the form of a docker container to ensure that your application works seamlessly in any environment.

Docker Container is a standardized unit which can be created on the fly to deploy a particular application or environment. It could be an Ubuntu container, CentOs container, etc. to full-fill the requirement from an operating system point of view. Also, it could be an application oriented container like Cake PHP container or a Tomcat-Ubuntu container etc. Figure 2 shows architecture of Docker.

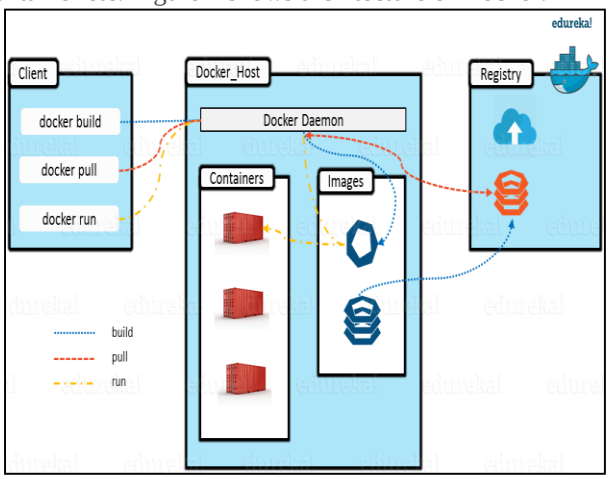


Fig. 2: Docker Architecture

Docker Architecture includes a Docker client which is used to trigger Docker commands, in which Docker Host is running the Docker Daemon and a Docker Registry for storing Docker Images. The Docker Daemon running within Docker Host is responsible for the images and containers. To build a Docker Image, CLI (client) is used to issue a build command to the Docker Daemon (running on Docker Host). The Docker Daemon will then build an image based on given inputs and save it in the Registry, which can be either Docker hub or a local repository. If one do not want to create an image, then just pull an image from the Docker hub, which would have been built by a different user. Finally, for creating a running instance of my Docker image, it is necessary to run command from the CLI, which will create a Docker Container.

Containers are the ready applications created from Docker Images or it is a running instance of a Docker Image and they hold the entire package needed to run the application. This happens to be the ultimate utility of Docker. Also Docker Registry is where the Docker Images are stored. The Registry can be either a user's local repository or a public repository like a Docker Hub allowing multiple users to collaborate in building an application. Even with multiple teams within the same organization can exchange or share containers by uploading them to the Docker Hub. Docker Hub is Docker's very own cloud repository similar to GitHub.

D. Problem Statement

Problem of effectively managing CPU utilization when many containers share a single set of resources. The problem is exacerbated in environments where an application is designed to maximize performance by utilizing resources as efficiently as possible. For example, in high-performance computing, applications are routinely optimized for cache access and locality. The performance of such applications can be destroyed in virtualization environments by removing the benefits of cache locality; moreover, stalls can result for synchronized processes or threads.

E. Motivation for Research Work

In recent years, lighter-weight virtualization solutions have begun to emerge in the computing landscape. These containerization solutions tend to rely on kernel-level implementations and to provide many of the benefits of virtual machines without the classical overhead inherent in virtualizing computing resources. The lower overhead, combined with simplicity of usage, leaves them primed to eventually take over much of landscape currently occupied by classical solutions. Because these solutions are still in their infancy, however, several research questions remain open in terms of how to effectively manage computing resources ranging from CPU utilization to network bandwidth. One such problem is how to effectively manage CPU utilization when many containers share a single set of resources. The problem is exacerbated in environments where an application is designed to maximize performance by utilizing resources as efficiently as possible.

For example, in high-performance computing, applications are routinely optimized for cache access and locality. The performance of such applications can be destroyed in virtualization environments by removing the

benefits of cache locality, also stalls can result for synchronized processes or threads. In order to minimize cache eviction and stalls, mechanisms are needed to carefully manage the resource use of containers. Here the actual focus is to provide a mechanism for containers to be scheduled in isolation. Specifically, by introducing the concept of time slicing currently missing at the container level. When such mechanism is used, only a single container with all its threads and processes is scheduled at any time on shared resources. By removing the scheduling overlap, minimizes the negative effects of oversubscribing the resources. Moreover, by allowing related threads to be scheduled at the same time, synchronization overhead is reduced due to scheduling variance.

F. Objective

The main objective of this research is to provide a container based algorithm on randomization and fuzzy inference algorithm to achieve efficient performance in heterogeneous of a processors power in cloud computing environment. The specific objectives of this research are, design a new algorithm that adopts the characteristics of randomization and Fuzzy Inference to make an efficient load balancing and covers their disadvantages, also implement the algorithm using live cloud environment and evaluate the algorithm system using response time metrics.

G. Scope

This research a hybrid load balancing algorithm is described which is mainly concentrate on overcoming the deficiencies in the performance of current algorithms. The limitations of this work are like, it considers the processor power factor; other specifications are out of scope, it focuses on normal arrival rate however other arrival rates are out of our scope, it is also a dynamic non-distributed load balancing algorithm which focuses only on improving scheduling performance in heterogeneous of a processors power in cloud computing environment. It is a local algorithm considers only one data center in one location, while the performance of such algorithm will be measured using real experiments.

II. BACKGROUND THEORY

Cloud computing is an information technology (IT) paradigm that enables ubiquitous access to shared pools of configurable system resources and higher-level services that can be rapidly provisioned with minimal management effort, often over the Internet. Cloud computing relies on sharing of resources to achieve coherence and economies of scale, similar to a public utility.

Third-party clouds enable organizations to focus on their core businesses instead of expending resources on computer infrastructure and maintenance. Advocates note that cloud computing allows companies to avoid or minimize up-front IT infrastructure costs. Proponents also claim that cloud computing allows enterprises to get their applications up and running faster, with improved manageability and less maintenance, and that it enables IT teams to more rapidly adjust resources to meet fluctuating and unpredictable demand. Cloud providers typically use a "pay-as-you-go" model, which can lead to unexpected operating expenses if administrators are not familiarized with cloud-pricing models.

Cloud computing exhibits some key characteristics such as, agility for organizations may be improved, as cloud computing may increase user's flexibility with re-provisioning, adding, or expanding technological infrastructure resources. Also cost reductions are claimed by cloud providers. A public-cloud delivery model converts capital expenditures (e.g., buying servers) to operational expenditure. This purportedly lowers barriers to entry, as infrastructure is typically provided by a third party and need not be purchased for one-time or infrequent intensive computing tasks. Pricing on a utility computing basis is "fine-grained", with usage-based billing options. As well, less in-house IT skills are required for implementation of projects that use cloud computing. The e-FISCAL project's state-of-the-art repository contains several articles looking into cost aspects in more detail, most of them concluding that costs savings depend on the type of activities supported and the type of infrastructure available in-house.

Device and location independence enable users to access systems using a web browser regardless of their location or what device they use (e.g., PC, mobile phone). As infrastructure is off-site (typically provided by a third-party) and accessed via the Internet, users can connect to it from anywhere. Maintenance of cloud computing applications is easier, because they do not need to be installed on each user's computer and can be accessed from different places (e.g., different work locations, while travelling, etc.).

Multitenancy enables sharing of resources and costs across a large pool of users thus allowing for centralization of infrastructure in locations with lower costs (such as real estate, electricity, etc.) and also peak-load capacity increases (users need not engineer and pay for the resources and equipment to meet their highest possible load-levels) utilization and efficiency improvements for systems that are often only 10–20% utilized.

Performance is monitored by IT experts from the service provider, and consistent and loosely coupled architectures are constructed using web services as the system interface. Resource pooling is the provider’s computing resources are commingle to serve multiple consumers using a multi-tenant model with different physical and virtual resources dynamically assigned and reassigned according to user demand. There is a sense of location independence in that the consumer generally have no control or knowledge over the exact location of the provided resource.

Productivity may be increased when multiple users can work on the same data simultaneously, rather than waiting for it to be saved and emailed. Time may be saved as information does not need to be re-entered when fields are matched, nor do users need to install application software upgrades to their computer. Reliability improves with the use of multiple redundant sites, which makes well-designed cloud computing suitable for business continuity and disaster recovery.

Scalability and elasticity via dynamic ("on-demand") provisioning of resources on a fine-grained, self-service basis in near real-time (Note, the VM startup time varies by VM type, location, OS and cloud providers), without users having to engineer for peak loads. This gives

the ability to scale up when the usage need increases or down if resources are not being used.

Security can improve due to centralization of data, increased security-focused resources, etc., but concerns can persist about loss of control over certain sensitive data, and the lack of security for stored kernels. Security is often as good as or better than other traditional systems, in part because service providers are able to devote resources to solving security issues that many customers cannot afford to tackle or which they lack the technical skills to address. However, the complexity of security is greatly increased when data is distributed over a wider area or over a greater number of devices, as well as in multi-tenant systems shared by unrelated users. In addition, user access to security audit logs may be difficult or impossible. Private cloud installations are in part motivated by users' desire to retain control over the infrastructure and avoid losing control of information security.

Hardware virtualization is the virtualization of computers as complete hardware platforms, certain logical abstractions of their componentry, or only the functionality required to run various operating systems. Virtualization hides the physical characteristics of a computing platform from the users, presenting instead an abstract computing platform. At its origins, the software that controlled virtualization was called a "control program", but the terms "hypervisor" or "virtual machine monitor" became preferred over time.

The term "virtualization" was coined in the 1960s to refer to a virtual machine (sometimes called "pseudo machine"), a term which itself dates from the experimental IBM M44/44X system. The creation and management of virtual machines has been called "platform virtualization", or "server virtualization", more recently. Platform virtualization is performed on a given hardware platform by host software (a control program), which creates a simulated computer environment, a virtual machine (VM), for its guest software. The guest software is not limited to user applications; many hosts allow the execution of complete operating systems. The guest software executes as if it were running directly on the physical hardware, with several notable caveats. Access to physical system resources (such as the network access, display, keyboard, and disk storage) is generally managed at a more restrictive level than the host processor and system-memory. Guests are often restricted from accessing specific peripheral devices, or may be limited to a subset of the device's native capabilities, depending on the hardware access policy implemented by the virtualization host. Virtualization often exacts performance penalties, both in resources required to run the hypervisor, and as well as in reduced performance on the virtual machine compared to running native on the physical machine.

III. METHODOLOGY AND EXPECTED OUTCOME

In a typical production environment, large groups of remote servers in a data center work under complex hierarchical layers with cross-domain cooperation. Resources are not only shared by multiple users, but are also dynamically re-allocated among containers on demand. Therefore, effective job scheduling, load balancing and resource allocation become critical to ensure the scalability and high availability of a virtualization computing environment.

In a CaaS framework, a container is the basic component that constitutes the business workflow, while a physical node (usually a server) is the fundamental carrier to deploy and execute containers. In practice, a cluster holds a large number of parallel workflows that execute independently. Each workflow varies on calculation time, data volume, network latency, submitted/completed time and other aspects, but they all share the resources in the same cluster. Therefore, it is difficult to accurately estimate the resource consumption for the entire workflow. For most static load balancing policies, the performance of the nodes is determined at the beginning of workflow execution, and this will cause all tasks in that workflow always been executed on the same node which it is assigned.

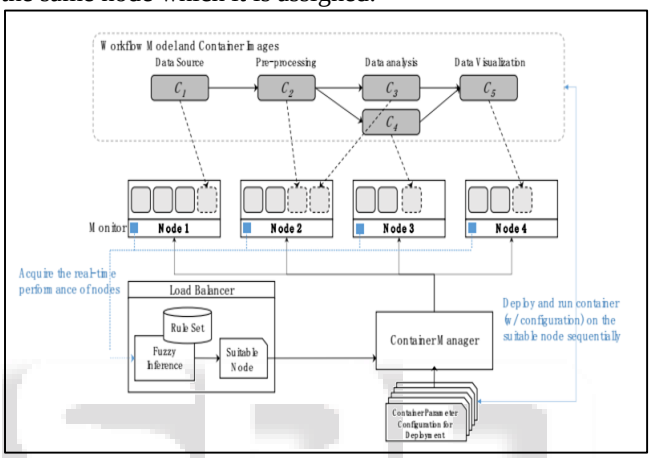


Fig. 3: Modules of resource allocation.

Conversely, a more reasonable policy is to break down the resource allocated to the granularity of the container, because a container dedicates to create separate units for every single application, service or backing resource. In this way, the workload is dynamically distributed among the nodes, in a more balanced manner, when it receives a request for a container deployment and execution. As shown in figure 3, two modules, load balancer (LB) and container manager (CM), coordinate to complete the resource allocation. LB identifies the least loaded physical node based on the current load status of each node in the cluster, and CM is responsible for deploying a container according to the corresponding configuration file generated in the previous stage. Every node in a cluster is a complex combination of multiple types of resources (e.g. CPU, memory, disks, networks and etc.), and the physical configurations of resources for each node may be heterogeneous as well. Similarly, to handle a complex system where a lot of uncertain parameters exist, for which fuzzy logic is applied to control such parameters again, instead of conventional modeling algorithms. The architecture of the LB for dynamic load balancer is shown in figure 4, including five components: status indicator collector, fuzzifier, rule base, inference engine, and the de-fuzzifier.

A. Data Collection

First, data about each node to determine the state of that node is needed. The collector collects data about each node from the /proc pseudo-file system, which contains the runtime

information about the current host system, e.g. system memory, devices mounted, hardware configuration and etc.

1) CPU usage

Various statistics about the CPU status can be obtained from /proc/stat, and we use the following formula to compute the final crisp value of CPU usage in percentage:

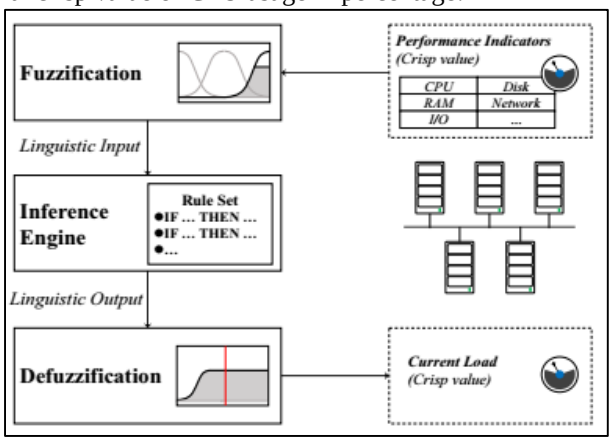


Fig. 4: Architecture of the LB for dynamic load balancer.

2) Mathematical Model

– Input:

Let 'S' be the

S= {D1, D2}

Where,

D1: {Set of Container Images}

D2: {Set of All Running Container Instances}

– Output:

O = {D, L, A}

D= {Container Details}

L= {log files}

A= {alerts}

– Functions:

S= {F1, F2, F3, F4}

F1= {Data collection}

F2= {Analysis}

F3= {Generation}

F4= {Storing}

– Success Conditions:-

Containers Stabilized

– Failure Conditions:-

Error message for failure

3) Mapping Diagram

f(r) be the rule which predicates work type provided that set of Input function

f(r) : f (I) → f (W) {many to one mapping}

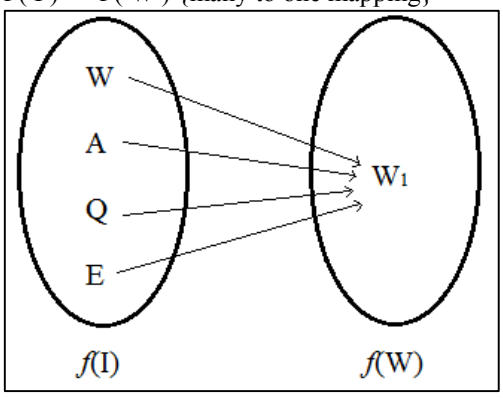


Fig. 1.3:

4) Functional Dependency

– Relational dependency is: Output function is dependent on Input attributes.
– Goal Function → Input attributes
– i.e. f(W) → f (I)

IV. IMPLEMENTATION STRATEGY

Docker is an application that makes it simple and easy to run application processes in a container, which are like virtual machines, only more portable, more resource-friendly, and more dependent on the host operating system. For a detailed introduction to the different components of a Docker container, check out The Docker Ecosystem: An Introduction to Common Components.

There are two methods for installing Docker on Ubuntu 16.04.

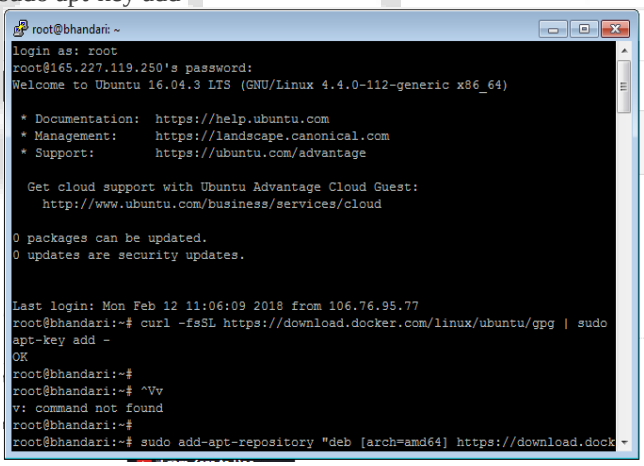
1) One method involves installing it on an existing installation of the operating system.
2) The other involves spinning up a server with a tool called Docker Machine that auto-installs Docker on it.

We are using how to install and use it on an existing installation of Ubuntu 16.04.

A. Installing Docker

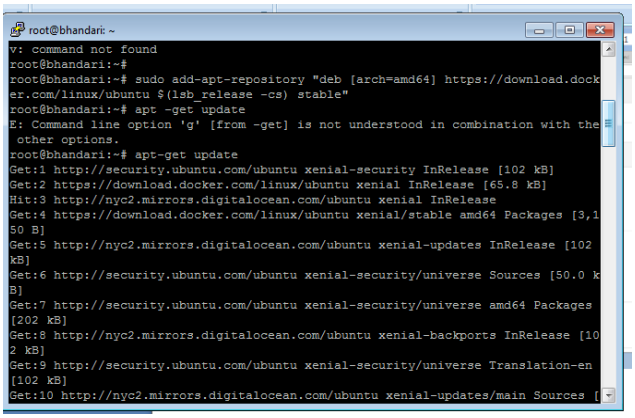
First, add the GPG key for the official Docker repository to the system:

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

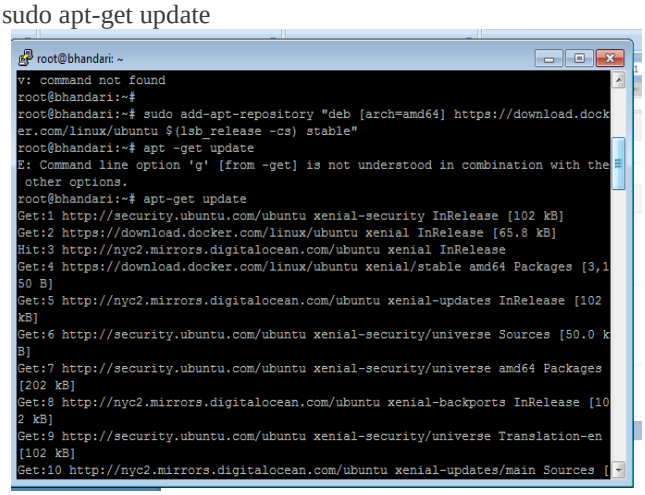


Add the Docker repository to APT sources

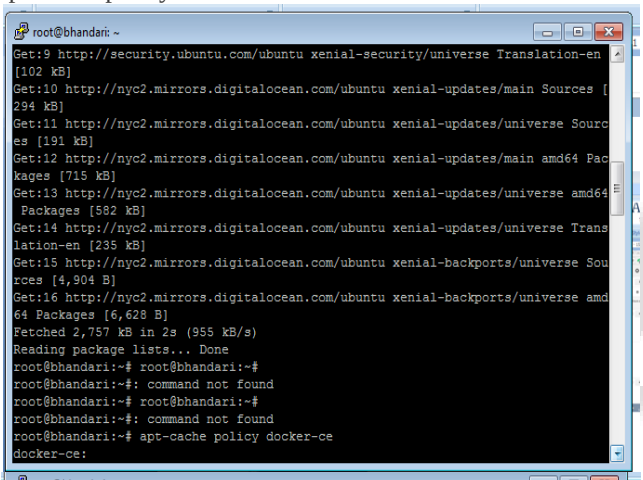
sudo add-apt-repository "deb[arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"

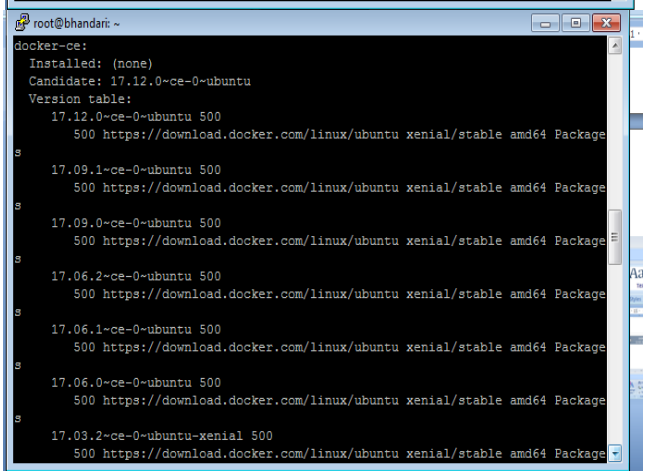


Next, update the package database with the Docker packages from the newly added repo:
sudo apt-get update



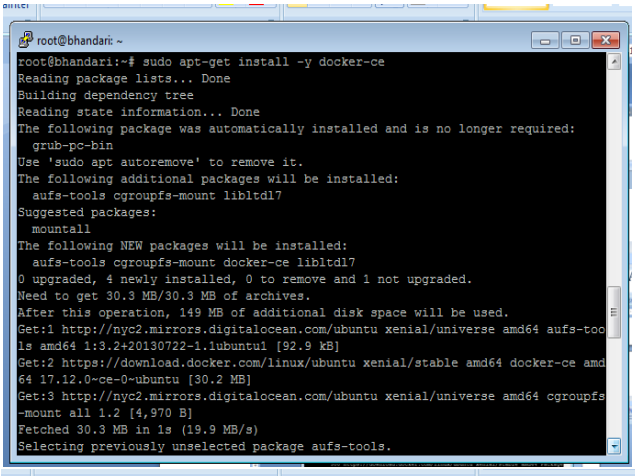
Make sure you are about to install from the Docker repo instead of the default Ubuntu 16.04 repo:
apt-cache policy docker-ce

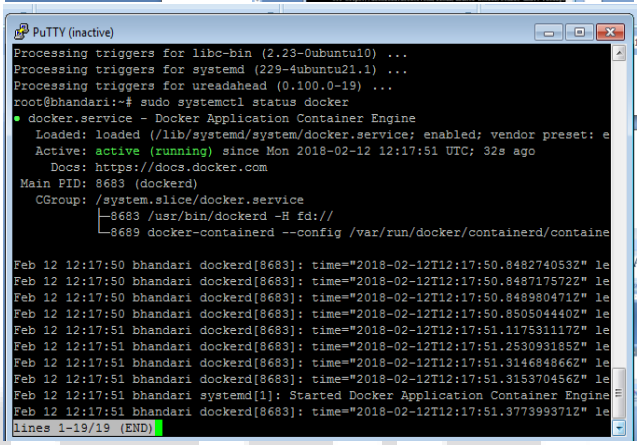




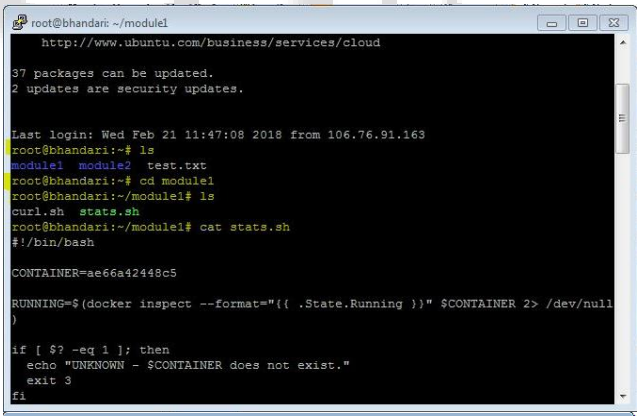
Notice that docker-ce is not installed, but the candidate for installation is from the Docker repository for Ubuntu 16.04. The docker-ce version number might be different.
Finally, install Docker:

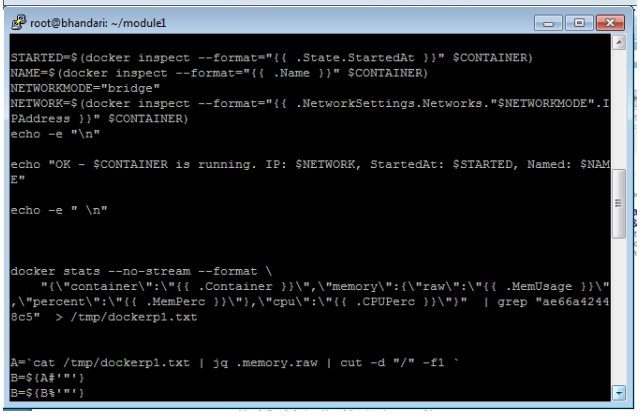
– sudo apt-get install -y docker-ce

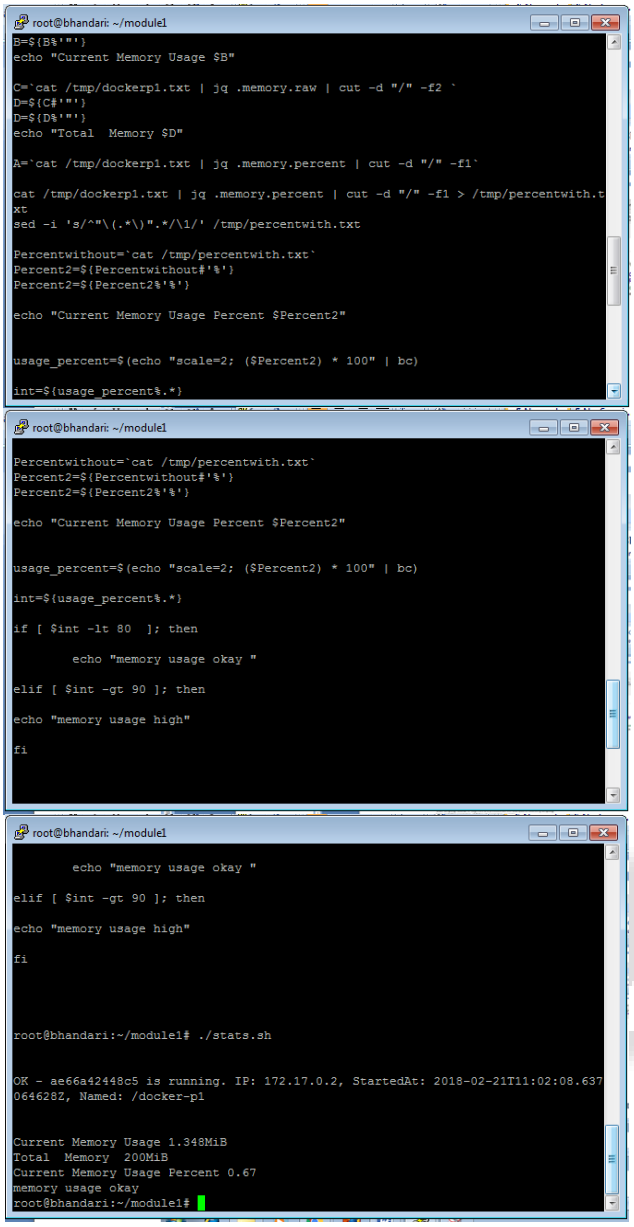




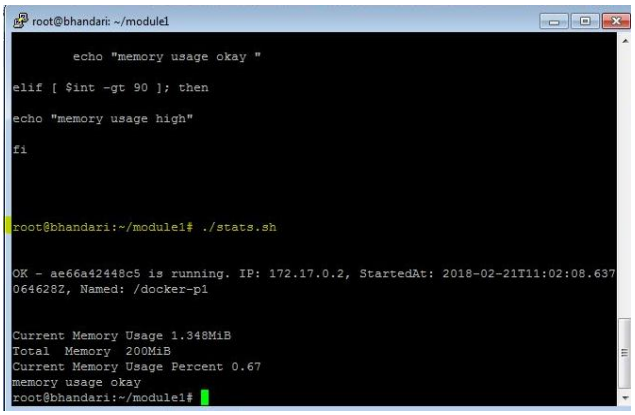
1) Module 1
ls commands to check list of contains/ list of directories.
Cd module1: changing over new directories
Cat stats.sh :create shell file







To run module 1 file use: ./stats.sh



REFERENCES

[1] X. W. Ye Tao, Xiaowei Xu and Guozhu Liu, "Dynamic Resource Allocation Algorithm for Container based Service Computing," IEEE 13th International Symposium on Autonomous Decentralized Systems, 2017.

[2] A. T. Saraswathi, Y. R. A. Kalaashri, and S. Padmavathi, "Dynamic Resource Allocation Scheme in Cloud Computing," Procedia Computer Science, vol. 47, pp. 30-36, 2015.

[3] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," Linux Journal, vol. 2014, p. 2, 2014.

[4] Sajid, M. and Z. Raza, "Cloud Computing: Issues & Challenges," in International Conference on Cloud, p. 35-41, 2013.

[5] Zhen Xiao, weijia song and Qi chen "Dynamic Resource allocation using Virtual Machines For Cloud Computing Environment", IEEE Transactions on parallel and distributed systems, ISSN: 1045-9219, vol. 24, No. 6, 2013.

[6] Ye Tao Xiaodong Wang, Xiaowei Xuin and Yinong Chen "Dynamic Resource Allocation Algorithm for Container-based Service Computing" in IEEE 13th International Symposium on Autonomous Decentralized Systems, 2017.

[7] J. Monsalve, A. Landwehr, and M. Taufer, "Dynamic CPU Resource Allocation in Containerized Cloud Environments," in IEEE International Conference on Cluster Computing (CLUSTER), pp. 535-536, 2015.

